intel®

APPLICATION NOTE

AP-224

September 1984

The BITBUS™ Interconnect: From Flight Simulation To Process Control, It Simplifies Distributed Intelligence

SHANKER MUNSHANI
RICHARD MCALISTER
PETER MACWILLIAMS

Reprinted from Solutions July/August 1984

Order Number 280072-001

APPLICATION NOTE

The BITBUS™ Interconnect: From Flight Simulation To Process Control, It Simplifies Distributed Intelligence

By Shanker Munshani, Richard McAlister and Peter MacWilliams

A large portion of microcontroller applications demand distributed modes of operation. Physically, this distribution can stretch from a few meters to several kilometers. The environment of operation varies from a very peaceful electrical environment to a very variable industrialized environment.

To accommodate changing application needs and technological advances, designers need a *flexible* interconnect for such systems that causes minimal impact to performance. Compatibility and the implementation of standards are key. Adhering to a standard has several advantages: designers of equipment need not waste time defining and testing their own standard, and end-users are more comfortable if the manufacturer has followed an industry standard. At the same time, another important feature is the capability of handling reliable communication activity without impacting CPU performance.

Intel's Distributed Control Modules (DCM) family accomplishes such goals for distributed applications. DCM defines an interconnect architecture and consists of:

The BITBUS interconnect – Interconnect serial control bus
iSBX™ 344 – BITBUS controller multimodule board
iRCB 44/10 – BITBUS remote controller board
iRMX™ 51 – Real-time multitasking executive
iRMX™ 510 – DCM support package
8044AH – 8-bit microcontroller with on-chip serial communication support

This application note will explain the structure and function of the BITBUS interconnect and explore its use in aircraft simulation and chemical process control.

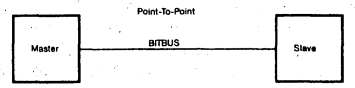
The BITBUS Interconnect: Rationale and Structure

To connect microcontrollers in a distributed application, two common approaches involve either building a custom interface and a custom cabling mechanism, or using other interfaces such as RS 232.

Yet, custom interfaces are faced with several disadvantages. They are generally very expensive, and a designer must design an interface in addition to designing the system. They also lack flexibility. For example, it is often impossible to add more input/output connections to a custom interface once it is implemented. Finally, custom interfaces pose problems for the end-user: they require considerable support; the cabling is generally cumbersome and slow; the distance over which they can be used is usually quite limited and their reliability may not be sufficient. Interfaces such as RS 232 are not an ideal alternative, either, because of the large amount of software support and cabling required.

The BITBUS interconnect avoids many of these problems. The BITBUS specification defines the data link protocol, message structure, protocol for a multitasking environment and a set of high-level commands for remote I/O access and application task control. This makes it very convenient to write high-level software interfaces. The BITBUS interconnect's high-level of definition means that the interface requirements can be implemented in silicon with minimal real estate at a low cost. This in turn reduces the complexity level.

The BITBUS in its simplest form is a pair of twisted wires. The BITBUS operates in a half duplex mode and can be used either in point-to-point operation or in a multi-drop environment. Figure 1 illustrates these two forms of connection. The BITBUS architecture supports a subset of the Synchronous Data Link Control (SDLC) protocol.



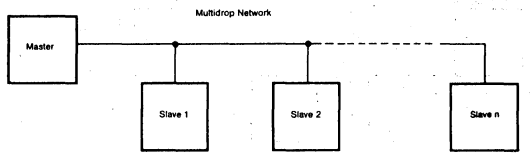


Figure 1.

There are three main objectives to be considered when using the BITBUS interconnect: speed of operation, distance over which communications has to take place, and number of nodes in the network. The BITBUS has two modes to meet these objectives: synchronous and self-clocked.

Synchronous mode

The synchronous mode is used for high speed operation. The distance over which this mode can be used is limited to 30 meters, and the number of modes in this set-up is restricted to 28 nodes. The speed of transmission in this mode is between 500 Kbits/sec to 2.4 Mbits/sec. To use this mode of operation, two pairs of twisted wires are required. One pair is used for the differential data clock signal (DCLK), while the other is used for the differential data signal (DATA). Figure 2 shows a typical synchronous mode interconnect.

Self-Clocked mode

In the self-clocked mode, as the name suggests, the clock is embedded in the data stream. In its simplest form, the

280072-001

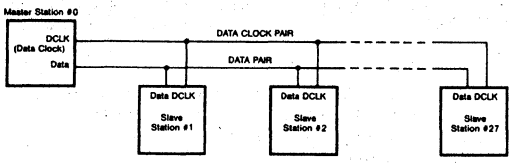


Figure 2. Synchronous Mode Network.

self-clocked mode requires just one pair of twisted wires. The speeds of operation in the self-clocked mode are 62.5 Kbits/sec and 375 Kbits/sec. The maximum distance of operation at 62.5 Kbits/sec is 1200 meters, and at 375 Kbits/sec is 300 meters. The maximum number of nodes in either case is 28. The self-clocked mode can be used to transmit over longer distances and to support more nodes by the use of repeaters. This, however, requires the use of an additional twisted pair of cables. This pair is used for Request To Send (RTS), which is the differential signal for transceiver control. The maximum number of repeaters allowed at 62.5 Kbits/sec are 10 and at 375 Kbits/sec are 2. Hence, at 62.5 Kbits/sec the distance over which the BITBUS link can be used is 13.2 kilometers, or 8.25 miles. The distance between the first node and the first repeater, the distance between two adjacent repeaters, and the distance between the last repeater and the last node are all called a segment. The maximum number of nodes permitted in any segment is 28, and the maximum number of nodes permitted in all the segments combined is 250. Figure 3 shows a self-clocked mode interconnect.

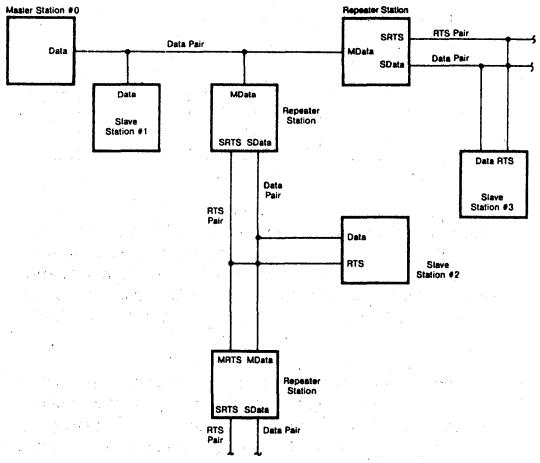


Figure 3. Self-Clocked Mode Interconnect.

The functions of the other parts of the BITBUS interconnect are described below:

iSBX 344 The iSBX 344 is a BITBUS controller multimodule board. This board can be used as either a master or a slave node in a BITBUS environment. This board has an iSBX connector and can be mounted on any iSBC board which has an iSBX connector and operates under any one of the following operating systems: iRMX 86, iRMX 286, iRMX 88, and ISIS-iPDS™ (Personal Development System). When the iSBX 344 multimodule board is used as a master node it is called a master extension, and when it is used as a slave node it is called a slave extension.

iRCB 44/10 The iRCB 44/10 board is a stand-alone BITBUS node. Unlike the iSBX 344 board, this board does not need a base board upon which to operate. The iRCB 44/10 board has a Eurocard single high-form factor and can be used as a stand-alone board. This board has 8 dedicated input lines and 16 programmable input/output lines.

iRMX 51 The iRMX 51 is a real-time, multitasking executive designed to monitor and control real-time events. A preconfigured version of the iRMX 51 Executive implements the BITBUS message format and provides all iRMX 51 facilities: task management, interrupt handling, and message passing.

iRMX 510 The iRMX 510 is a package of software aids to interface MULTIBUS® and iPDS ISIS systems to BITBUS systems in both run-time and development environments. It provides a simple software interface for iRMX 86, 88, 286 and iPDS ISIS operating systems compatibility. It provides a means for inexpensive remote control and communication in MULTIBUS-based systems.

8044AH The 8044AH with the DCM firmware provides the basic BITBUS interface. The 8044AH integrates a high performance 8-bit microcontroller, the Intel 8051 core, with an intelligent/high performance serial communication controller, called the Serial Interface Unit. The on-chip ROM can be used for the DCM firmware.

By virtue of these products, support for the BITBUS interconnect comes at various levels. The 8044AH chip with the DCM firmware provides the designer with the facility to integrate the BITBUS into the system at the very lowest level. Alternatively, the iSBX 344 multimodule board with an iSBX connector can be plugged into a system design at the highest level. Since the BITBUS interconnect is intelligent, it is capable of handling reliable communication activity with minimal interaction with the host processor.

Setting up a BITBUS Network

Figure 4 shows a typical BITBUS network. iPDS, Intel's Personal Development System, can be used as a master station to control the BITBUS network. The iPDS is a stand-alone development system with a CRT, a keyboard and a 5-1/4" floppy disk drive. The iSBX 344 board can act as a master extension on the iPDS base-processor board. This master station is capable of controlling up to 249 slave stations in a multi-drop fashion. The iSBX 344 is numbered as station #0 and is connected via the BITBUS to station 1, which is an iRCB 44/10 board. The BITBUS is then used to connect to slave station number 2, which is another iRCB 44/10 board. From here the BITBUS is routed to an iSBX 344 board mounted on an iSBC 86/30 board, a MULTIBUS-based board. This is termed as station #4. (Note there is no station #3; the station numbering does not have to follow a sequential order). The BITBUS then routes over to station #9, an iSBX 344 board on an iRCB 44/10 board. After this the BITBUS travels to another iRCB 44/10 board, station #10. From here the BITBUS goes to station #15, which is an iRCB 44/10 board with an analog multimodule

280072-001

board. Thus, there is one master station number 0, and six slave stations with the following numbers: 1, 2, 4, 9, 10, 15.

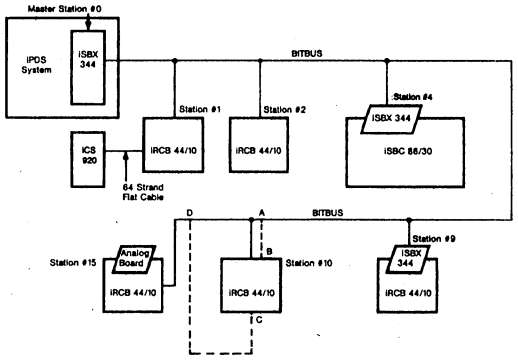


Figure 4.

If the distance from station 0 to 15 is less than 30 meters, this network can operate in either the synchronous mode or self-clocked mode. Assume the distance between stations 0 and 10 to be 200 meters, between stations 10 and 15 to be 250 meters and the speed required for the operation of the network to be 375 Kbit/sec. Since the maximum distance of a segment at 375 Kbits/sec is 300 meters, a repeater must be placed in the network. Since the iRCB 44/10 has on-board repeaters, station 10 could serve the function of a repeater. If this is the case, the BITBUS route then follows the direction of ABCD, as opposed to AD, as was the case in the previous example. Station 10 exists as a slave station and also as a repeater. Thus, the network has two segments, each less than 300 meters long. As a result, the network will work in asynchronous mode at 375 Kbits/s. It should be noted that stub lengths play an important role in a multi-drop network. Stub length is the distance from the drop point on the network to the node. Care should be taken to keep this as small as possible.

Creating a Task

Each individual station is now set up as an individual node. Since each BITBUS interconnect is an intelligent node, each node has its own tasks. Each station can have up to eight tasks. The Remote Access and Control (RAC) task is designated as task 0, so there can be seven more user-defined tasks. Using the same set-up as in figure 4, assume station 1 has only task 0 and no other user-defined tasks. This station could be used to perform any of the RAC Access or Control functions. A simple example could be to write a set of 1's to an output port and then flip this value to 0's. This could be achieved by using the EXT_IO_Write RAC function to write a set of 1's to an output port and then using the EXT_OI_XOR function to flip these bits. If an iCS 920 Digital Signal Conditioning board is connected to the output port of the iRCB 44/10 board, the LEDs (Light-Emitting Diodes) on that port of the iCS 920 board will turn on and off. If this task is run at a station, this will cause the LEDs to flash continuously. This demonstrates the simplicity of the I/O capabilities.

Message Structure The iRMX 51 Executive allows tasks to interface with one another via a simple message-passing facility.

Link: is a 2-byte field used by the executive.

Message_Length: is a byte value specifying the number of bytes in the message. This is 7 bytes of header information plus the number of bytes of user data. The maximum message size is configurable.

Message_Type: is a bit that determines whether this is an order message (=0) or a reply to a message (=1). If it is an order, the nucleus will use the consumer address as the destination. If it is a reply it will use the producer address as the destination.

Src_ext: is a bit value which indicates whether the sending task of an order message is located on an extension (=1) or on a device (=0).

Dest_ext: is a bit value which indicates whether the receiving task of an order message is located on an extension (=1) or on a device (=0).

Trk: is a bit field used during BITBUS transfer for tracking the message. Trk is set to 0 before sending an order message.

Station_address: For messages delivered locally (on the same chip), this field is 0. For messages delivered over a parallel interface only, this field is OFFH. For order types of messages, to be delivered from a master device or its extension to a slave device or its extension, this field is the SDLC station address of the slave device.

Source_task_id: is a byte value containing the task i.d. for the message originator. Upon reply, this value is interpreted as the reply destination.

Destination_task_id: is a byte value containing the task i.d. for the message destination. Upon reply, this value is interpreted as the reply source.

Command/response: is a byte field which is available for use by the sending and receiving tasks. It can be used for sending command or reply information. This field has pre-defined functions when communicating with the RAC function.

Message_information: is a user-defined field following the 7 bytes of message header information. For messages destined for the RAC task, this area has a fixed structure.

280072-001

Considering the example mentioned above to flash LEDs, the message sent and received would be (in Hex):

Message Sent	00	00	0B	40	01	00	0C	C2	FF	C0	FF
Message Received	XX	XX	0B	C0	01	00	00	C2	FF	C0	FF
									00		00

The first two bytes are the Link field. This field is reserved. The next byte specifies the message length, which is 7 bytes of header information plus 4 bytes of user-defined message. Therefore, the total message length is 11 bytes (i.e., OBH).

In the next byte, the first bit is set to 0 to indicate an order type and the second bit is set to 1, since the order message resides on an extension (the iSBX 344 is on an extension). The third bit is set to 0, since the task which receives the order message resides on a device (iRCB 44/10). The fourth bit is always set to zero before sending a message. The last four bits are reserved and set to 0's. This byte in binary is then equal to 01000000, i.e., 40H. In the received message the only field changed is the first bit, because now this bit is a reply and hence changes to 1. The received byte in binary is therefore equal to 11000000, i.e., COH.

The next byte defines the slave station address. Since the slave station address was 1, this field and its reply field are both 01.

The next byte is broken into two nibbles. Since the sending task and the receiving tasks were both RAC tasks (Task 0), this field is 00. (Note: this is not strictly the case for extensions).

Since the RAC order message has been generated, this field selects the RAC service for that message. The EXT_10_XOR RAC function has the value OCH. Thus the value at the ports defined will be Exclusive-ORed.

The last four bytes of the message follow the following format: address byte, followed by the data byte, followed by the address byte, and so on. The first byte (C2H) defines the address of the output port. The next byte (FFH) is the value written to this port (C2H). The next byte (C0H) is the address of another output port, followed by the byte value (FFH) written to this port (C0H). The received message has the same value in this field during the write operation. When the value is XOR the data field values change to 00H.

An Aircraft Application

Flight simulation uses the capabilities of the BITBUS interconnect. Figure 5 shows an implementation for flight simulation. As the figure demonstrates, flight simulation can be broken down into a block diagram level consisting of six sections, namely:

1) Pitch: This section is responsible for the vertical movements of the aircraft. The inputs required for this section

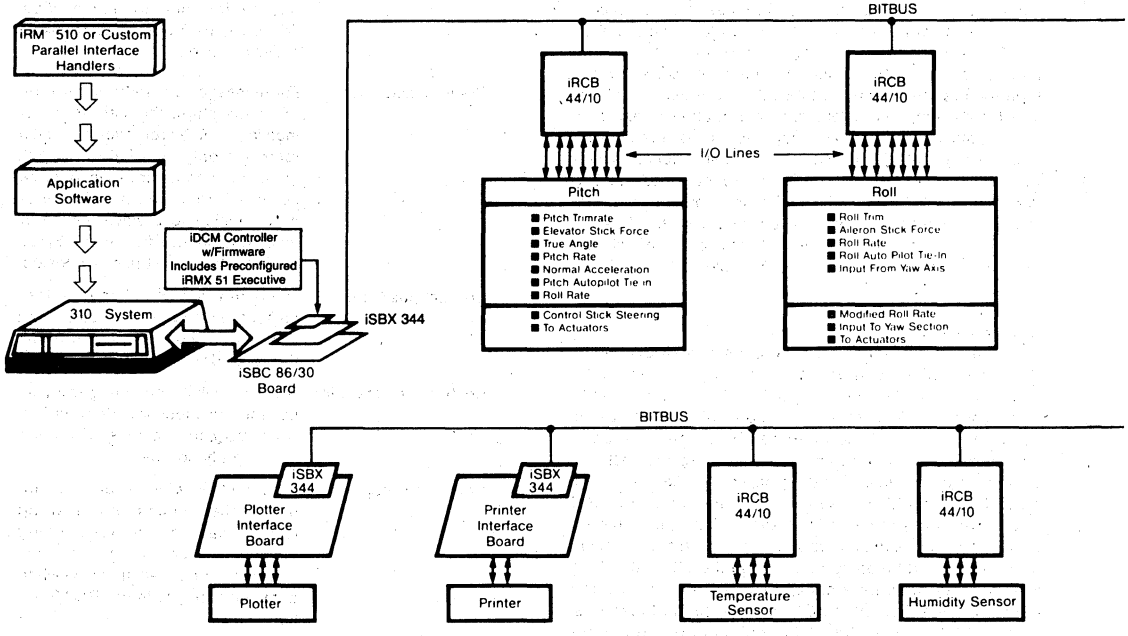


Figure 5.

280072-001

are pitch trim rate, elevator stick force, true angle, pitch rate, normal acceleration, pitch autopilot tie-in, and roll rate. The outputs of this section are: control stick steering and input to the actuators.

2) Roll: The Roll section is responsible for the rolling movement of the aircraft about its belly. The inputs required for this section are: roll trim, aileron stick force, roll rate, roll autopilot tie-in, and input from yaw axis. The outputs of this section are the modified roll rate, input to the yaw section, and input for the actuators.

3) Yaw: This section is responsible for the horizontal movements of the aircraft. The inputs for the yaw section are: yaw trim, rudder pedal force, yaw rate, lateral acceleration and yaw axis. The outputs of this section are input to the aileron rudder interconnect and input to the actuators.

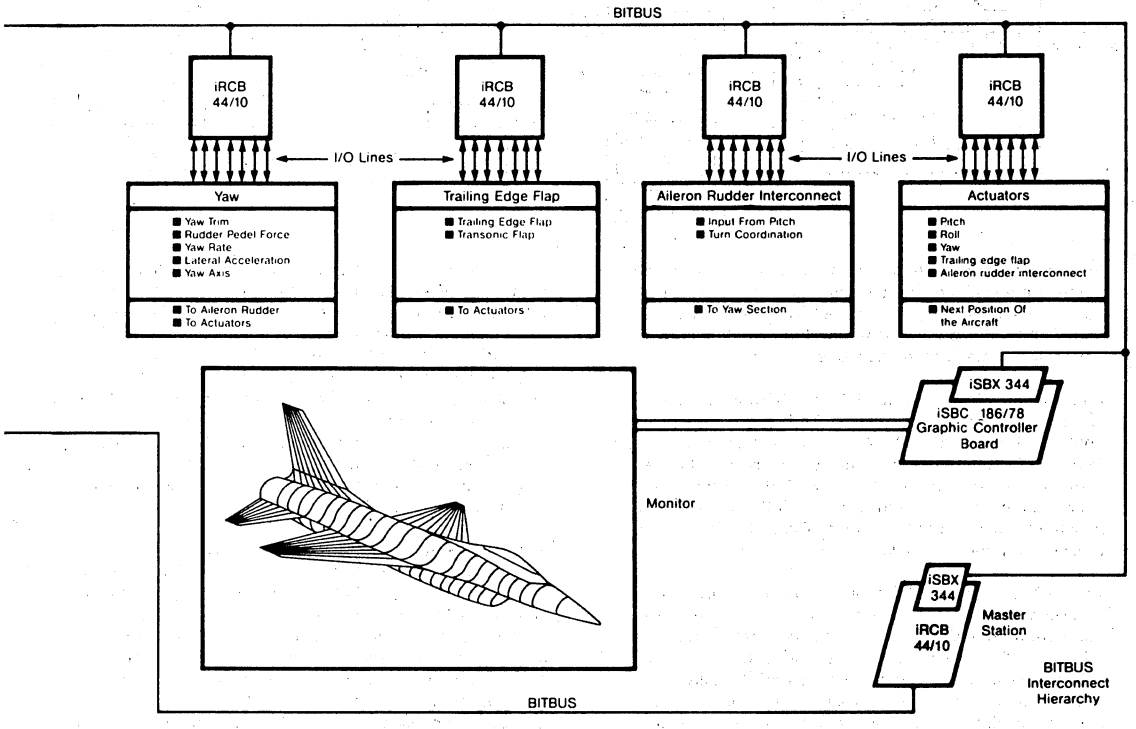
4) Trailing Edge Flap: The trailing edge flap section is responsible for the drag on the aircraft, mainly during takeoff and landing. The inputs to this section are: trailing edge flap command, and transonic flap. The output of this section is input to the actuators.

5) Aileron Rudder Interconnect: When an aircraft rolls, its center of gravity shifts. Therefore, a force is required to counteract the gravitational force in order to keep the aircraft stable. This is achieved by the Aileron Rudder Interconnect. The inputs for this section are input from the pitch and turn coordination. The output of this section is input to the yaw section.

6) Actuators: Actuators are basically transducers which constantly monitor the aircraft. Their inputs are the various forces and factors currently acting on the aircraft and their outputs are the command signals for the next position of the aircraft.

In this example, Intel's 310 system is used as the master station. This is achieved by plugging an iSBX 344 board onto the iSBC 86/30 board inside the 310 system. The iSBX 344 board provides the BITBUS interconnect. Each section of the aircraft block diagram is controlled via an iRCB 44/10 board. (Depending on the device used to take the measurements and the accuracy desired, several iRCB 44/10s could be used in one section.) With this simple insertion, the BITBUS can be used to monitor each section. The actuators are also connected via the BITBUS, providing the control mechanism.

The BITBUS model monitors sections in the following manner: Each node (iRCB 44/10) has several tasks (a maximum of 8) residing on it. These tasks monitor the various parameters in each section. I/O ports on the iRCB 44/10 can be used to read the value of the different parameters in each section. They then perform the necessary computation and return the output parameters of that section to the master node. For example, in the "Pitch section", there would be a task to read the input port which is connected to a sensor monitoring the pitch trim rate. Similarly, there would be a task to compute the output of the pitch sec-



280072-001

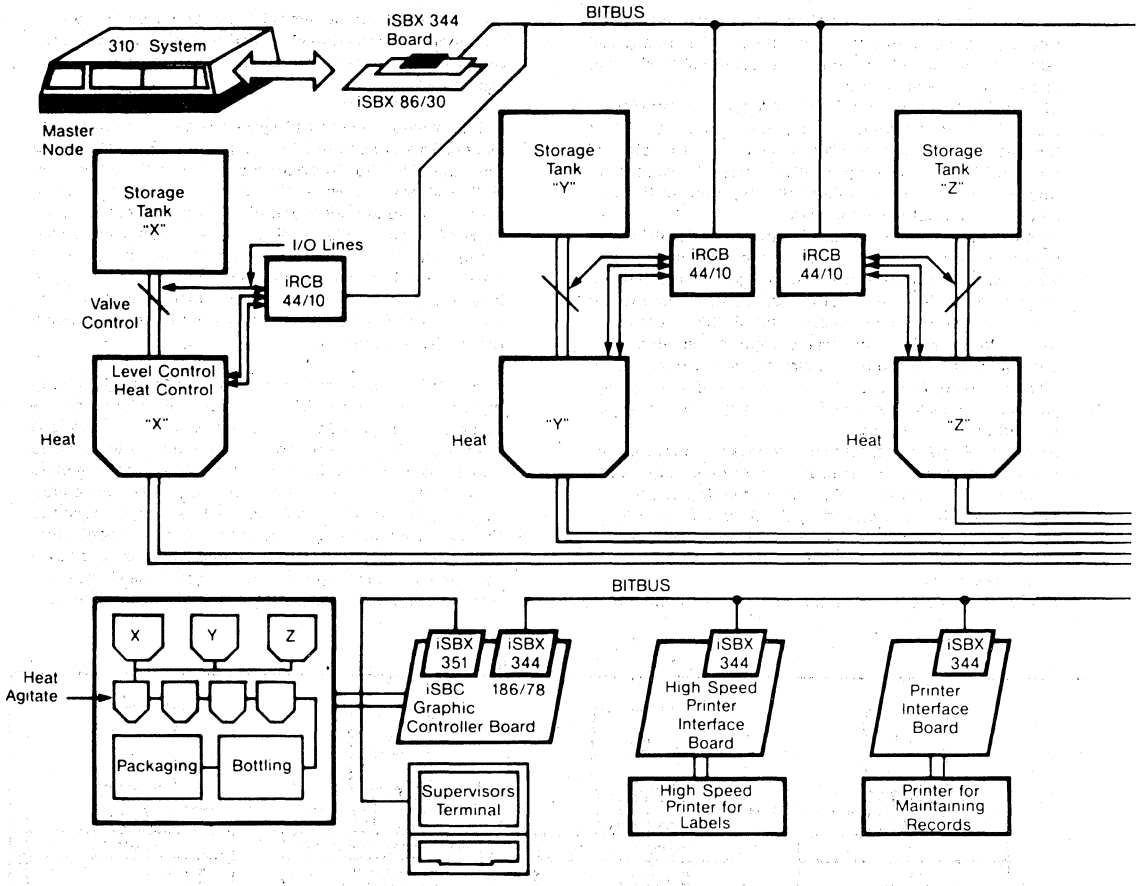


Figure 6.

tion and either write this value to an output port which controls a transducer, or send this value to the master so as to be used as an input to the actuators. In this manner, the BITBUS interconnect controls all the input and output parameters in each of the sections.

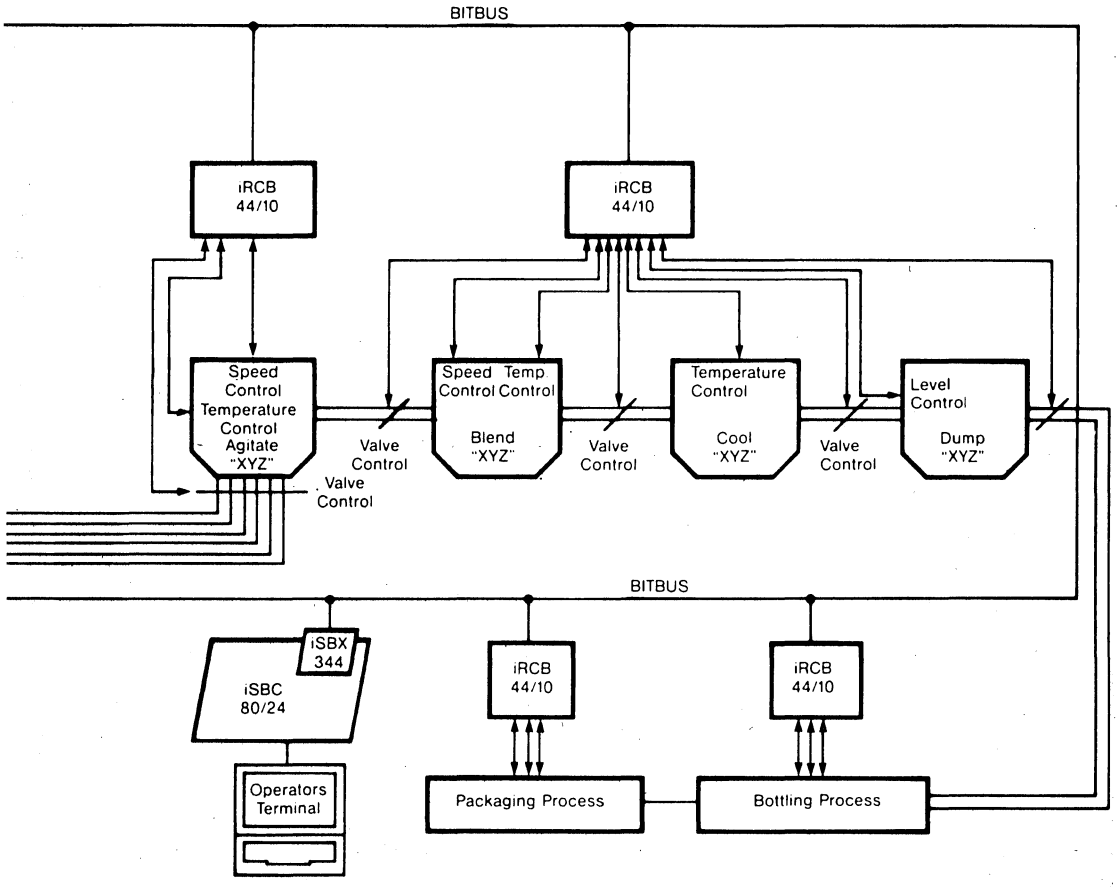
An iSBC 186/78 board is tied into the BITBUS network via an iSBX 344 board. The iSBC 186/78 board is a graphics controller board. The iSBX 344 board is just another node in the BITBUS network, and it helps in conveying the message from the master node to the iSBC 186/78 board. The master node sends messages to the iSBC 186/78 to display the simulations of the aircraft. The iSBC 186/87 then presents a graphical display on a CRT.

The BITBUS interconnect is defined to provide a high speed serial control bus for hierarchical systems. In many instances there are several slow devices or devices that do not need prime attention from the master node in a BITBUS network. This is a good reason to use the hierarchical facility of the BITBUS interconnect. In this example, the iSBX 344 multimodule residing on the iRCB 44/10 board uses this hierarchical interconnect. This iRCB 44/10 is now the master node for the four nodes that control the printer, the plotter, the temperature sensor, and the humidity sensor. The plotter logs the position of the aircraft at one minute intervals and the printer records the weather conditions every three minutes. The temperature and humidity sensors are tied into the I/O ports of the iRCB 44/10 nodes. These slave nodes continuously monitor the readings, and at the end of every 3-minute duration find the average value. These average values are then sent to the hierarchical master node, which in turn sends these values to the printer to log the values. The hierarchical master at the end of every minute receives the aircraft's position information from the main master and feeds this information to the plotter.

The BITBUS Approach to Process Control

Process control is another example where distributed intelligence is important. Figure 6 shows a simple process control flow. In this example, three chemicals, namely 'X', 'Y', and 'Z', are used to produce a product 'XYZ'. A 310 system is the master node. A slave node, in this case, an iRCB 44/10 board, is tied to each chemical unit at the start of the process. The I/O capabilities of this node control the flow of the chemical from the storage tank and the level of the chemical in the heating tank. Once the chemical reaches the required level in the heating tank, this node

280072-001



also closes the storage tank valve. After closing this valve, this node then turns the heater on in the tank and controls its temperature. At the end of the heating period, it opens the valve to the next tank.

Another node at the Agitate tank monitors the activities of this tank. This node controls the flow of the chemicals into the tank, the temperature of the chemicals, and the speed of the agitation motor.

Another node controls all the Blend, Cool and Dump stages. In the blend tank, the node controls the speed, temperature and the flow into and out of the tank. In the cooling tank, temperature and flow are controlled. The dump tank control monitors the level of the final product in this tank. If it reaches a near-full stage, it sends a message to the master station which then either stops the process momentarily or else diverts the action onto another dump tank. The BITBUS network then goes on to control the assembly line by controlling the bottling process and the packaging process. The same network is also used to log the packaged product information onto a line printer. Another node could be used to control a high-speed printer which would print labels with the batch number, the date of manufacture and the expiration date. If desired, an iSBX 344 node could be connected to an iSBC 186/78 board, which would run a color monitor in a supervisor's office, giving the supervisor a pictorial view of the entire manufacturing line. The BITBUS set up could also accommodate operators having a node at their benches to do any form of human interaction that is desired.

Conclusion

The BITBUS interconnect is capable of handling reliable communication activity without impacting CPU performance. It is a low-cost, high-performance approach that is easy to use. It does not require expensive cabling or special cables. It provides intelligent I/O capabilities. It has several speeds of operation in two modes and can be used over long distances at comparable speeds. The flexibility of the BITBUS interconnect makes it very attractive, since more slave nodes can be added with minimal effort. It is intended to be an important tool in an industrial environment, and, by virtue of its open architecture and standardized implementation, to continue to be of use as application needs evolve over time.

280072-001